Contrôle continu - Compilation (L3 Info)
Durée : 2h - documents interdits

Conventions

— Axiome : symbole non terminal de la partie gauche de la première production de la grammaire
— Symboles non terminaux : lettres *MAJUSCULES ITALIQUES*
— Symboles terminaux : lettres `minuscules true-type` ou caractères spéciaux simples

1 Vrai ou faux *(4 pts)*

Pour chacune des affirmations suivantes dites si elle est vraie ou fausse en justifiant brièvement et de façon convaincante votre réponse (seules les réponses accompagnées d'une justification seront prises en compte).

Q1.1 Certaines grammaires ambiguës sont LR. (1pt)
Faux.
Une grammaire G est ambigue s'il existe au moins un mot de $L(G)$ auquel G associe plus d'un arbre de dérivation ou, ce qui revient au même, plusieurs dérivations droites.
Une grammaire G est LR si sa table LR ne comporte aucun conflit et donc associe à tout mot de $L(G)$ une seule dérivation droite.
Une grammaire ne peut donc pas vérfier les deux propriétés ci-dessus.

Q1.2 Tout langage hors-contexte peut être reconnu par un automate à pile non déterministe. (1pt)
Vrai.
Nous avons vu en cours comment construire à partir d'une grammaire G un automate à pile non déterministe reconnaissant $L(G)$

Q1.3 Le langage des expressions régulières sur l'alphabet $\{a, b\}$ est un langage hors-contexte. (1pt)
Vrai.
Voici un exemple de grammaire (ambiguë) reconnaissant le langage qui nous intéresse :
$E \rightarrow E.E|E + E|E * |(E)|a|b$

Q1.4 Les automates à pile reconnaissant les grammaires ambiguës sont non déterministes. (1pt)
Vrai
Soit G une grammaire et A un automate gauche reconnaissant $L(G)$. Pour tout mot $m \in L(G)$, A permet de constuire une dérivation gauche menant à m. Si G est ambigüe, il existe au moins un mot $m' \in L(G)$ auquel G associe au moins deux dérivations gauches. Pour pouvoir produire plus d'une dérivation gauche pour m', A doit être non déterministe.

2 Écriture de grammaire *(6 pts)*

En C, les variables doivent être déclarées avec leurs types. Une variable en C peut être de type simple, comme un entier (`int`), un réel (`float`) ou un caractère (`char`) ; ou de type complexe comme une `struct` ou une `union`.[1] De plus, tous ces types peuvent devenir des pointeurs si le nom de la variable est précédé d'une étoile `*`. Les types complexes contiennent des listes de champs qui sont, à leur tour, des déclarations de variables simples ou complexes, et ainsi de suite récursivement. Voici un exemple :

1. La différence entre une `struct` et une `union` est que les champs d'une `struct` représentent une conjonction de variables alors que celles d'une union représentent une disjonction de variables.

```
struct personne {
    struct { char *prenom;
             char *nom; } identite;
    char sexe;
    int age;
    union { float salaire;
            float chomage; } revenus;
} pers1;
```

La variable `pers1` est de type complexe `struct`. Elle contient des champs simples tels que l'âge (`int`) et le sexe (`char`), et des champs complexes tels que l'identité (`nom` et `prenom` dans une `struct`) et les revenus (soit un `salaire`, soit des indemnités de `chomage` dans une `union`). On peut utiliser les champs de la variable `pers1` dans des expressions ou instructions, par exemple, `pers1.revenus.chomage = 1045.56`.

Vous devez écrire une grammaire hors contexte non ambiguë qui reconnaît une liste de déclarations de variables de types simples et complexes en C. Nous nous limitons aux types simples `int`, `char`, `float`, aux pointeurs et aux types complexes `struct` et `union`. Chaque déclaration se termine par un point-virgule. [2] Les identificateurs (noms) de variables du langage sont reconnus par l'analyseur lexical. Utilisez le terminal `IDENTIF` pour les représenter dans votre grammaire.

Q2.1 Écrivez une grammaire non ambiguë qui reconnaît une liste de déclaration de variables de type simple. Ignorez pour le moment les pointeurs et les types complexes.

```
LDS -> DS LDS | eps
DS -> TS LI ;
TS -> int | float | char
LI -> IDENTIF LI'
LI' -> , IDENTIF | eps
```

Q2.2 Écrivez une grammaire non ambiguë qui reconnaît une déclartion de variable de type complexe `struct` ou `union` contenant une liste de variables simples ou complexes. Ignorez pour le moment les pointeurs.

```
DC -> TC IDENTIFO { LD } IDENTIF ;
IDENTIFO -> INDENTIF | eps
TC -> struct | union
LD -> DS LD | DC LD | eps
```

Q2.3 Écrivez la grammaire non ambiguë complète qui mélange les deux grammaires ci-dessus et qui reconnaît une liste de déclarations de types simples, pointeurs et types complexes qui peuvent être combinés récursivement.

```
LD -> D LD | eps
D -> DS | DC
DS -> TS LI ;
TS -> int PTRO | float PTRO | char PTRO
PTRO -> * | eps
DC -> TC IDENTIFO { LD } LI ;
IDENTIFO -> INDENTIF | eps
TC -> struct PTRO | union PTRO
LI -> IDENTIF LI'
LI' -> , IDENTIF | eps
```

2. Nous ne reconnaîtrons pas les déclarations multiples `int valeur1, valeur2, valeur3;` ni les initialisations `int i = 0`.

3 Premier, Suivant, dérivation, ambiguïté *(4 pts)*

Soit la grammaire G_1, d'axiome A :

$$\begin{array}{lcl} A & \rightarrow & B\ C\ D\ A \mid \varepsilon \\ B & \rightarrow & \texttt{a} \mid \texttt{b} \\ C & \rightarrow & \texttt{c} \mid \varepsilon \\ D & \rightarrow & \texttt{a} \mid \varepsilon \end{array}$$

Q3.1 Calculez Premier et Suivant pour les symboles non terminaux de G_1.

(2pt)

	Premier	Suivant
S	$\langle$	$\perp$
L	$\langle\ \varepsilon$	$\perp$
E	$\langle$	;
M	0 1	$\rangle$

Q3.2 Donnez une dérivation droite du mot `aaaa`. (1pt)

Q3.3 Démontrez que la grammaire G_1 est ambiguë. (1pt)

4 Analyse SLR *(6 pts)*

Soit la grammaire G_2 :

$$\begin{array}{llcl} 1 & A & \rightarrow & B\ A \\ 2 & A & \rightarrow & \varepsilon \end{array} \qquad \begin{array}{llcl} 3 & B & \rightarrow & \texttt{a}\ C \\ 4 & B & \rightarrow & \texttt{c} \end{array} \qquad \begin{array}{llcl} 5 & C & \rightarrow & \texttt{b}\ C \\ 6 & C & \rightarrow & \varepsilon \end{array}$$

Q4.1 Construisez l'automate $LR(0)$ de G_2 à l'aide des fonctions Fermeture et Aller_à

```
>> Terminals:
{ A, B, C }

>> Nonterminals:
{ a, b, c }

>> Productions:
a -> b a
   | eps

b -> A c
   | C

c -> B c
   | eps

>> Start symbol:
a
```

```
######################################################################

First(a) = eps C A
First(b) = C A
First(c) = eps B

######################################################################

Follow(a) = '$'
Follow(b) = C A '$'
Follow(c) = C A '$'

######################################################################

Canonical LR(0) item sets (kernel items above --- dashed line)

I0: {
  aBis -> • a
  ----
  a -> •
  a -> • b a
  b -> • A c
  b -> • C
}
I1: {
  b -> A • c
  ----
  c -> •
  c -> • B c
}
I2: {
  b -> C •
  ----
}
I3: {
  aBis -> a •
  ----
}
I4: {
  a -> b • a
  ----
  a -> •
  a -> • b a
  b -> • A c
  b -> • C
```

```
}
I5: {
  c -> B • c
  ----
  c -> •
  c -> • B c
}
I6: {
  b -> A c •
  ----
}
I7: {
  a -> b a •
  ----
}
I8: {
  c -> B c •
  ----
}

######################################################################

Automaton's GOTO transitions

GOTO( I0, A) -> I1
GOTO( I0, C) -> I2
GOTO( I0, a) -> I3
GOTO( I0, b) -> I4
GOTO( I1, B) -> I5
GOTO( I1, c) -> I6
GOTO( I4, A) -> I1
GOTO( I4, C) -> I2
GOTO( I4, a) -> I7
GOTO( I4, b) -> I4
GOTO( I5, B) -> I5
GOTO( I5, c) -> I8

######################################################################

SLR ACTION/GOTO table

State 0 :
  ACTION : '$' -> r a -> eps
  ACTION : A -> s1
  ACTION : C -> s2
  GOTO   : a -> 3
```

```
  GOTO   : b -> 4

State 1 :
  ACTION : '$' -> r c -> eps
  ACTION : A -> r c -> eps
  ACTION : B -> s5
  ACTION : C -> r c -> eps
  GOTO   : c -> 6

State 2 :
  ACTION : '$' -> r b -> C
  ACTION : A -> r b -> C
  ACTION : C -> r b -> C

State 3 :
  ACTION : '$' -> acc

State 4 :
  ACTION : '$' -> r a -> eps
  ACTION : A -> s1
  ACTION : C -> s2
  GOTO   : a -> 7
  GOTO   : b -> 4

State 5 :
  ACTION : '$' -> r c -> eps
  ACTION : A -> r c -> eps
  ACTION : B -> s5
  ACTION : C -> r c -> eps
  GOTO   : c -> 8

State 6 :
  ACTION : '$' -> r b -> A c
  ACTION : A -> r b -> A c
  ACTION : C -> r b -> A c

State 7 :
  ACTION : '$' -> r a -> b a

State 8 :
  ACTION : '$' -> r c -> B c
  ACTION : A -> r c -> B c
  ACTION : C -> r c -> B c
```

Q4.2 Étant donné les ensembles SUIVANT de G_2 ci-dessous, construisez la table SLR de G_2

	PREMIERS	SUIVANTS
A	{ ε, a, c}	{ $ }
B	{ a, c}	{ a, c, $}
C	{ ε, b}	{ a, c, $}

(2pt)

5 Analyse SLR *(2 pts)*

Soit la grammaire G_3 ci-dessous et sa table SLR correspondante :

1 $A \rightarrow B\ A$
2 $A \rightarrow \varepsilon$
3 $B \rightarrow C\ D$
4 $C \rightarrow$ a
5 $C \rightarrow$ b
6 $D \rightarrow$ c
7 $D \rightarrow \varepsilon$

	a	b	c	$	A	B	C	D
0	d1	d2		r2	3	4	5	
1	d2	r4	r4	r4				
2	d2	r5	r5	r5				
3				acc				
4	d1	d2		r2	6	4	5	
5	r7	r7	d7	r7				8
6				r1				
7	r6	r6		r6				
8	r3	r3		r3				

Q5.1 Simulez l'analyse LR du mot abc. Les configurations sont représentées par le triplet (α, w, y), où α est la pile avec le sommet à droite, w est la partie de la bande de lecture qui commence au caractère sous la tête de lecture (ce qui reste à analyser), et y est la séquence de symboles de sortie (numéros des productions appliquées lors des réductions). (1pt)

($0,abc$,ε) $\vdash$
($01,bc$,ε) $\vdash$
($05,bc$,4) $\vdash$
($058,bc$,47) $\vdash$
($04,bc$,473) $\vdash$
($042,c$,473) $\vdash$
($045,c$,4735) $\vdash$
($0457,$,4735) $\vdash$
($0458,$,4735) $\vdash$
($046,$,47353) $\vdash$
($03,$,473531) $\vdash$